instruction manual

i!-EquipmentMonitor

integration!Solutions

Software Limited Agreement

LIMITED WARRANTY

LIMITED WARRANTY. AMX Corporation warrants that the SOFTWARE will perform substantially in accordance with the accompanying written materials for a period of ninety (90) days from the date of receipt. Any implied warranties on the SOFTWARE and hardware are limited to ninety (90) days and one (1) year, respectively. Some states/countries do not allow limitations on duration of an implied warranty, so the above limitation may not apply to you.

CUSTOMER REMEDIES. AMX Corporation's entire liability and your exclusive remedy shall be, at AMX Corporation's option, either (a) return of the price paid, or (b) repair or replacement of the SOFTWARE that does not meet AMX Corporation's Limited Warranty and which is returned to AMX Corporation. This Limited Warranty is void if failure of the SOFTWARE or hardware has resulted from accident, abuse, or misapplication. Any replacement SOFTWARE will be warranted for the remainder of the original warranty period or thirty (30) days, whichever is longer.

NO OTHER WARRANTIES. AMX Corporation disclaims all other warranties, either expressed or implied, including, but not limited to implied warranties of merchantability and fitness for a particular purpose, with regard to the SOFTWARE, the accompanying written materials, and any accompanying hardware. This limited warranty gives you specific legal rights. You may have others which vary from state/country to state/country.

NO LIABILITY FOR CONSEQUENTIAL DAMAGES. In no event shall AMX Corporation be liable for any damages whatsoever (including, without limitation, damages for loss of business profits, business interruption, loss of business information, or any other pecuniary loss) arising out of the use of or inability to use this AMX Corporation product, even if AMX Corporation has been advised of the possibility of such damages. Because some states/countries do not allow the exclusion or limitation of liability for consequential or incidental damages, the above limitation may not apply to you.

U.S. GOVERNMENT RESTRICTED RIGHTS

The SOFTWARE and documentation are provided with RESTRICTED RIGHTS. Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of The Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 or subparagraphs (c)(1) and (2) of the Commercial Computer Software--Restricted Rights at 48 CFR 52.227-19, as applicable. Manufacturer is AMX Corporation, 3000 Research Drive, Richardson, TX 75082.

If you acquired this product in the United States, this Agreement is governed by the laws of the State of Texas.

Should you have any questions concerning this Agreement, or if you desire to contact AMX for any reason, please write: ***AMX Corporation, 3000 Research Drive, Richardson, TX 75082***.

Table of Contents

Introduction 1

Supported Operating Systems 1
Minimum PC Requirements 1
Installing i!-EquipmentMonitor 2

Running i!-EquipmentMonitor 3

Sending Email 3
Receiving Email 4
Configuring for Timezone 6
i!-EquipmentMonitorOut.axi 7
Constants 7
Structures 8
Variable 8
Functions 8
i!-EquipmentMonitorIn.axi 11
Constants 11
Structures 11
Variables 11
Functions 12

Introduction

i!-EquipmentMonitor™ is an application that allows you to send and receive e-mail directly from a NetLinx™ Control System. i!-EquipmentMonitor is primarily used to send and receive NetLinx Control System e-mails, such as sending e-mail notifications for system problems or equipment trouble and receiving
e-mails for Control System messaging. i!-EquipmentMonitor consists of the following files:

- **i!-EquipmentMonitorIn.axi** is an include file for receiving e-mails using POP3 protocol.
- **i!-EquipmentMonitorOut.axi** is an include file for sending e-mails using SMTP protocol.
- **i!-EquipmentMonitorTest.axs** is a test program using both i!-EquipmentMonitorIn.axi and i!-EquipmentMonitorOut.axi.

Supported Operating Systems

- Windows 95®/98® (with at least 48 MB of installed memory)
- Windows NT 4.0® Workstation or Server (service pack 6 B or greater, with at least 64 MB of installed memory)
- Windows 2000® Professional or Server (running on a Pentium 233 MHz processor (minimum requirement); 300 MHz or faster recommended, with 96 MB of installed memory.)

Minimum PC Requirements

- Windows-compatible mouse (or other pointing device)
- At least 5 MB of free disk space (150 MB recommended)
- VGA monitor, with a minimum screen resolution of 800 x 600
- A Network adapter
- A Web server such as Personal Web Server (PWS) or Internet Information Server (IIS)
  - Windows 95® and 98® use PWS.
  - Windows 2000® Professional or Server, and Windows NT 4.0® Server use IIS.

Installing i!-EquipmentMonitor

1. In Explorer, double-click **i!-EquipmentMonitorSetup.exe** from the directory window where you downloaded the i!-EquipmentMonitor install program.
2. After reading the License Agreement, select **I Agree** and **Next** to proceed.
3. The Welcome To i!-EquipmentMonitor Setup dialog appears, reminding you to close all Windows programs before going any further. Click **Next** to proceed.
4. In the Select i!-EquipmentMonitor Install Location dialog, use the Browse button to navigate to a directory other than the default install directory, if desired. Click **Next**.
5. In the i!-EquipmentMonitor Shortcut Creation dialog, select **Install Shortcut Icons** for the installed components on your desktop, if desired.
6. Click **Next** in the Start i!-EquipmentMonitor Installation dialog to install the selected components.

The program prompts you to restart your system to complete the installation.

Running i!-EquipmentMonitor

Very little work is required to add e-mail support to your existing NetLinx code. Receiving and sending e-mail are independent of each other; each one will be covered in it's own section. You do not need to add support for sending and receiving if only one of the features is needed.

Sending Email

To support sending email, first include the i!-EquipmentMonitorOut.axi (page 7) into your program:

```
#INCLUDE 'i!-EquipmentMonitorOut.axi'      // Include to send email
```

Next, make sure that the default IP local port used by i!-EquipmentMonitorOut.axi is available on your system. i!-EquipmentMonitorOut.axi uses local port 0:10:0 for sending emails. Make sure there is no current entry in your `DEFINE_DEVICE` section for 0:10:0. If there is a current entry for 0:10:0, you can change the existing entry to another local port number or override the default local port used by
i!-EquipmentMonitorOut.axi like this in the `DEFINE_DEVICE` section:

```
dvSmtpSocket  =  0:3:0
```

Next, you need to initialize the SMPT server value by calling `SmptSetServer()`. You need the name or IP address of your local SMPT server, which you can obtain from you Network administrator. Using a name for the server is acceptable if you have DNS properly configured on the NetLinx Master. Otherwise, you need an IP address. Make sure to use the SMPT server value here. Often, the "email server" refers to the POP3 server; most likely, this is not what you need. Once you have the correct SMPT server name or address, call `SmptSetServer()` like this:

```
SmtpSetServer('smtpserver.mydomain.com')
```

or

```
SmtpSetServer('192.168.12.175')
```

If the SMTP email server requires user authentication to send email, you must configure i!-EquipmentMonitor with the username and password of a valid account registered to the SMTP server. You can do this by calling the SmtpSetUser() function. This line is typically included immediately after the SmtpSetServer() initialization command function. An example is provided below:

SmtpSetUser('user1','password') // include only if you need SMTP authentication

If the SMTP email server allows anonymous access and does not require authentication, you simply comment out the previous SmtpSetUser() function call. Without an SMTP username and password configured, i!-EquipmentMonitor will connect to the SMTP server with anonymous access.

Now, all you need to do is call the function that sends an email. If you want to send an email every time someone presses a button on a touch panel, your code would look like this:

```
BUTTON_EVENT[dvTP,1]
{
  PUSH:
  {
    SmtpQueMessage('fromAddress@mydomain.com',
                                    'toAddress@mydomain.com ',
                                    'My Emails subject',

     'this is the body of my email!  Wow, this is Cool!',
                                    '')
  }
}
```

The call to `SmtpQueMessage()` causes your email to queue and transmit to the SMTP server. The maximum number of emails that can be queued is controlled by the constant `SMTP_MAX_EMAILS`, which defaults to 10. You can override this value if you choose; you will probably never have a need to since the emails are sent very quickly.

The parameters to `SmtpQueMessage()` control where your email will be sent. The first parameter is the From Email Address. This usually does not have to be a real email address but some SMTP server configurations may require a valid one. It is best to obtain an email address from your email administrator.

The second parameter is the To Email Address. This can be a list of addresses separated by a ";" so you can send an email to more than one recipient in a single call to `SmtpQueMessage()`. The next two parameters are the subject and message of the email. The message can be up to `SMTP_MSG_MAX` characters long (the default value is 2000 but you can override it if necessary).

The last parameter is an attachment file name. If you supply a value for this parameter, i!-EquipmentMonitorOut attempts to open this file from the Master's file system and include it as an attachment to the email. Binary files are not supported at this time so the file must be ASCII (text) only. If the file does not exist or cannot be opened, an error is printed to the terminal, but the email is still sent without an attachment.

Receiving Email

To support receiving email, first include the i!-EquipmentMonitorIn.axi (page 11) into you program:

```
#INCLUDE 'i!-EquipmentMonitorIn.axi'       // Include to get email
```

Next, make sure that the default IP local port used by i!-EquipmentMonitorIn.axi is available on your system.

i!-EquipmentMonitorIn.axi uses local port 0:11:0 for sending emails. Make sure there is no current entry in your `DEFINE_DEVICE` section for 0:11:0. If there is a current entry for 0:11:0, you can change the existing entry to another local port number or override the default local port used by i!-EquipmentMonitorIn.axi like this in the `DEFINE_DEVICE` section:

```
dvPop3Socket  =  0:4:0
```

Next, you need to initialize the POP3 server value by calling `Pop3SetServer ()`. You will need the name or IP address of your local POP3 server, provided by your Network administrator. Using a name for the server is acceptable if you have DNS properly configured on the NetLinx Master. Otherwise, you will need an IP address. Once you have the correct POP3 server name or address, call Pop3SetServer like this:

```
Pop3SetServer ('mail.mydomain.com')
```

Or

```
Pop3SetServer ('192.168.12.175')
```

Next, you need to setup the user and password for the email account you will be retrieving email from. Your email administrator should supply you with a user name and password for an email account that can receive email. Once you have these, call `Pop3SetUser()` and supply these values like this:

```
Pop3SetRefresh(300,1)                  // How often the check email server in
Seconds and should I delete?
```

Supplying a refresh time of 0 seconds disables automatic email retrieval. If you decide you want to retrieve email manually, all you need to do is call `Pop3GetEmail()`. `Pop3GetEmail()` takes 1 parameter: a flag (1 or 0) indicating if you want email deleted from the server. You can call `Pop3GetEmail()` even if you have setup for automatic email retrieval to force email to be retrieve. You might supply the user with a button to force email to be retrieved like this:

```
BUTTON_EVENT[dvPanel,6]
{
  PUSH:
    Pop3GetEmail(0);
}
```

You can check for the email to arrive by waiting for the offline message from the `dvPop3Socket` device. i!-EquipmentMonitor makes the emails available to you in three different variables: `sPop3EmailMessage`, `nPop3QtyMail` and `nPop3TotalMail`. `nPop3QtyMail` and `nPop3TotalMail` tell you how many email messages were retrieved (up to `POP3_MAX_EMAILS`, default = 20) and how many emails were on the server when i!-EquipmentMonitor last logged in. Up to `POP3_MAX_EMAILS` are downloaded and all other emails, if any, remain on the server until you retrieve them again.

`sPop3EmailMessage` is an array of structures containing the actual emails. The structure contains the following items:

`LmsgSize`	The number of bytes in the email message.
`cFrom[]`	A string containing the senders email address.
`cFromPersonal[]`	The friendly name of the sender (if one was supplied).
`cTo[]`	A string containing the recipient's email address or addresses.
`cToPersonal[]`	The friendly name of the recipient if one was supplied.
`cDate[]`	A string containing the data and time the email was sent.
`cSubject[]`	The subject of the email.
`cMessage[]`	The body of the email.
`NattachCount`	The number of attachments to the email.
`cAttachments[][]`	The names of the files attached.

The subject and body are the items you need most in the structure. The count of file attachments tells you the total number of files attached but the `cAttachments` containing up to `POP3_ATTACH_MAX` (default is 5) file names. The attached files are not saved. Only the file names are supplied for reference.

Continued ↓

You can use the following code to loop through the downloaded emails whenever new email arrives:

```
DATA_EVENT[dvPop3Socket]
{
  OFFLINE:
  {

    STACK_VAR
    INTEGER nLoop
    Integer nLoop1
    For (nLoop=1;nLoop<=nPop3QtyMail;nLoop++)
    {
      SEND_STRING 0,'i!Email Test-Print Message'
      SEND_STRING 0,' '
      SEND_STRING 0,"'Message #',Itoa(nLoop)"
      SEND_STRING 0,"'Date:',sPop3EmailMessage[nLoop].cDate"
      SEND_STRING 0,"'From: "',sPop3EmailMessage[nLoop].cFromPersonal,
                                    '" <',sPop3EmailMessage[nLoop].cFrom,'>'"
      SEND_STRING 0,"'To: "',sPop3EmailMessage[nLoop].cToPersonal,

                                    '" <',sPop3EmailMessage[nLoop].cTo,'>'"
      SEND_STRING 0,"'Subject:',sPop3EmailMessage[nLoop].cSubject"
      SEND_STRING 0,"'Message:',sPop3EmailMessage[nLoop].cMessage"
      SEND_STRING
0,"'Attachments:',Itoa(sPop3EmailMessage[nLoop].nAttachCount)"
      For (nLoop1=1;nLoop1<=sPop3EmailMessage[nLoop].nAttachCount;nLoop1++)
        SEND_STRING 0,"'Attachment ',Itoa(nLoop1),':',

sPop3EmailMessage[nLoop].cAttachments[nLoop1]"

      SEND_STRING 0,' '
    }
  }
}
```

Once the emails are processed, you can delete any emails you like by calling `Pop3ClearEmailMessage ()` or `Pop3ClearAllEmailMessages ()`. `Pop3ClearEmailMessage` allows you to delete one email at a time; `Pop3ClearAllEmailMessages ()` allows you to delete all the emails at once.

Configuring for Timezone

The i!-EquipmentMonitorOut.axi file can read the time zone information from i!-TimeManager and includes this information in email and notifications. Simply include the i!-TimeManager moodule and make sure to name the i!-TimeManager virtual device '`vdvTmEvents`'. The file '**i!-EquipmentMonitorTest with i!-TimeManager.axs**' provides an example of using these two applications together.

Using i!-TimeManager is recommended for use with i!-EquipmentMonitor since some email clients may improperly display the time when the email or notification was sent. i!-TimeManager provides i!-EquipmentMonitor with a universal time reference, including any Daylight Savings time offsets, and includes this information in the email or notification.

*The i!-TimeManager Module is not included with i!-EquipmentMonitor. To obtain the i!-TimeManager Module (**i!-TimeManager.tko**), please download the i!-TimeManager install from our web site.*

i!-EquipmentMonitorOut.axi

Constants

The following table lists i!-EquipmentMonitorOut.axi constants.

i!-EquipmentMonitorOut.axi Constants	
dvSmtpSocket	The IP device number for sending e-mails (default = 0:10:0}.
SMTP_VERSION	The version number of the include file.
SMTP_PORT	IP Port that the SMTP server is listening on (default = 25).
SMTP_SERVER_TO	Timeout in 1/10 for contacting the SMTP server (default = 1200).
SMTP_URL_MAX	Maximum length for e-mail server name (default = 1000).
SMTP_USER_MAX	Maximum length for e-mail addresses (default = 500).
SMTP_LINE_MAX	Maximum length for date, subject and attached file (default = 256).
SMTP_MAX_EMAILS	Maximum length for number of queued e-mails (default = 10).
SMTP_MSG_MAX	Maximum length for e-mail message (default = 2000).

Continued ▼

Structures

The following defines an i!-EquipmentMonitorOut.axi structure:

```
Structure _sSMTPMessage
{
  CHAR cDate[SMTP_LINE_MAX];
  CHAR cSource[SMTP_USER_MAX];
  CHAR cDest[SMTP_USER_MAX];
  CHAR cSubject[SMTP_LINE_MAX];
  CHAR cMessage[SMTP_MSG_MAX];
  CHAR cFile[SMTP_LINE_MAX];
}
```

Variable

The following is an i!-EquipmentMonitorOut.axi variable:

```
VOLATILE
CHAR
bSMTPDebug    Set to 1 to enable debugging
```

Functions

The following are a list of i!-EquipmentMonitorOut.axi functions:

i!-EquipmentMonitorOut.axi Functions	

SmtpQueMessage
Call this to send an e-mail message.
	Syntax:
	`SLONG SmtpQueMessage(CHAR Source[],CHAR Dest[],CHAR Subject[],CHAR Message[],CHAR File[])`
	SmtpQueMessage has these arguments:
	Source: String containing the senders e-mail address.
	Dest: String containing the recipients e-mail address or addresses.
	Subject: String containing the subject of the e-mail.
	Message: String containing the message body of the e-mail.
	File: String containing the ASCII (text) only file name to attach to the e-mail.
	SmtpQueMessage returns these values:
	-1: If the message was not successfully queued for sending.
	>0: If the message was successfully queued for sending.
	Example:
	`SmtpQueMessage('me@mydomain.com','vmorrison@moondance.com', 'Wild Nights', 'Are they calling?','')`
	Remarks:
	`SmtpQueMessage` should be called if you want to send a message. The To parameter can contain multiple addresses separated by a ";". The file parameter is the path and file of an ASCII (text) only file contained on the master's file systems. This file is transmitted as an attachment.

i!-EquipmentMonitorOut.axi Functions (Cont.)	
SmtpSetTimeOffset(CHAR Offset[]) Call this to configure the local timezone	SmtpSetTimeOffset has these arguments: Offset String containing the local timezone offset. This string is formatted as "+/-HHMM" where "+/=" is "+" or "-" depending on your timezone relative to GMT, "HH" is the offset in hours relative to GMT and "MM" is the offset in minutes relative to GMT. Some Common Offsets are: -0500 : Eastern time (UTC - 5:00) -0600 : Central time (UTC - 6:00) -0700 : Mountain time (UTC - 7:00) -0800 : Pacific time (UTC - 8:00) -0900 : Alaska (UTC - 9:00) -1000 : Hawaii (UTC - 10:00) +0000 : Greenwich Mean Time (same as UTC) +0000 : Dublin, Edinburgh, Lisbon, London (UTC + 0:00) +0100 : Brussels, Copenhagen, Madrid, Paris, Vilnius (UTC + 1:00) +0100 : Amsterdam, Berlin, Bern, Rome, Stockholm, Vienna (UTC + 1:00) +0200 : Eastern Europe (UTC + 2:00) +0900 : Osaka, Sapporo, Tokyo (UTC + 9:00) +0800 : Hong Kong SAR (UTC + 8:00) +0700 : Bangkok, Hanoi, Jakarta (UTC + 7:00) +0300 : Baghdad, Kuwait, Riyadh (UTC + 3:00) +0200 : Israel (UTC + 2:00) -0600 : Mexico City, Tegucigalpa (UTC - 6:00) -0600 : Central America (UTC - 06:00) +0200 : Jerusalem (UTC + 2:00) +0300 : Nairobi (UTC + 3:00) Example: `SmtpSetTimeOffset ('-0600')` Remarks: SmtpSetTimeOffset should be called to configure i!-EquipmentMonitor to send emails with the correct time. Some SPAM detectors may mark an e-mail as spam if the timezone is not correctly reported. i!-EquipmentMonitor is designed to work with i!-TimeManager to obtain timezone information. If you have included i!-TimeManager in your program and the virtual device for i!-TimeManager is defined as "`vdvTmEvents`", the timezone will be configured correctly.

SmtpSetServer
Sets your SMTP Server Name for your use.
	Syntax:
	`SmtpSetServer(CHAR Server[])`
	SmtpSetServer has these arguments:
	Server: String containing the name or IP of your e-mail (SMTP) server.
	SmtpSetServer does not return a value.
	Example:
	`SmtpSetServer('mail.amx.com')`
	Remarks:
	`SmtpsetServer` should be called in `DEFINE_START` of your application.

i!-EquipmentMonitorOut.axi Functions (Cont.)	
SmtpSetUser Call this to configure the username and password for SMTP server authentication to send outbound emails.	SmtpSetUser(CHAR LogInName[],CHAR LogInPass[]) SmtpSetUser has these arguments: LogInNameString containing the username for the SMTP server. LogInPassString containing the password for them STMP server. SmtpSetUser doe not return any values. Example: `SmtpSetUser ('MyUserName','MyPassword')` Remarks: SmtpSetUser should be called to configure i!-EquipmentMonitor to send emails to an SMTP server that required authentication.
EncrBase64Encode This function is used internally to encrypt the username and password for SMTP server authentication.	You should not have to call this function directly. To configure SMTP authentication, please see the SmtpSetUser() function.

ConfigNotify
Sets your notification paramaters for your use.
	Syntax:
	`ConfigNotify(CHAR Source[],CHAR Dest[],CHAR Subject[])`
	ConfigNotify has these arguments:
	Source — String containing the senders email address.
	Dest — String containing the recipients email address or addresses.
	Subject — String containing the subject of the email.
	ConfigNotify does not return a value.
	Example:
	`ConfigNotify('user2@test.com','user1@test.com','Equipment Notification Room 301')`
	Remarks:
	ConfigNotify should be called in `DEFINE_START` of your application. You must also call the SmtpSetServer Function.
	The To parameter can contain multiple addresses separated by a ";".

SendNotify
Call this function to send an equipment notification.
	Syntax:
	`SLONG SendNotify(CHAR Message[],CHAR File[])`
	SendNotify has these arguments:
	Message — String containing the message body of the email.
	File — String containing the ASCII (text) only file name to attach to the email.
	SendNotify returns these values:
	-1 — If the message was not successfully queued for sending.
	>0 — If the message was successfully queued for sending.
	Example:
	`SendNotify('The VCR needs to be cleaned.','')`
	Remarks:
	SendNotify should be called if you want to send a notification. The To, From and Subject used in the ConfigNotify function is included in the notification.
	The file parameter is the path and file of an ASCII (text) only file contained on the Master's file systems. This file is transmitted as an attachment.

Continued

i!-EquipmentMonitorIn.axi

Constants

The following table lists the i!-EquipmentMonitorIn.axi constants.

i!-EquipmentMonitorIn.axi Constants	
dvPop3Socket	The IP device number for sending e-mails (default = 0:11:0}.
POP3_VERSION	Version number of the include file.
POP3_PORT	IP Port that the POP3 server is listening on (default = 110).
POP3_BUFFER_MAX	Maximum size of buffer for IP socket (default = 2048).
POP3_SERVER_TO	Timeout in 1/10 for contacting the POP3 server (default = 1200).
POP3_URL_MAX	Maximum length for e-mail server name (default = 1000).
POP3_USER_MAX	Maximum length for e-mail addresses (default = 500).
POP3_PASS_MAX	Maximum length for e-mail password (default = 100).
POP3_LINE_MAX	Maximum length for date, subject and attached file (default = 256).
POP3_MAX_EMAILS	Maximum number of e-mails to be retrieved (default = 20).
POP3_MSG_MAX	Maximum size of message body (default = 2000).
POP3_ATTACH_MAX	Maximum number of attachment file names stored (default = 5).

Structures

The following describes an i!-EquipmentMonitorIn.axi structure:

```
Structure _sEmailMessage
{
  Integer   nMsgNum;
  Char      cUniqueID[POP3_USER_MAX];
  long      lMsgSize;
  Char      cFrom[POP3_USER_MAX];
  Char      cFromPersonal[POP3_USER_MAX];
  Char      cTo[POP3_USER_MAX];
  Char      cToPersonal[POP3_USER_MAX];
  Char      cDate[POP3_USER_MAX];
  Char      cSubject[POP3_LINE_MAX];
  Char      cMessage[POP3_MSG_MAX];
  Integer   nAttachCount;
  Char      cAttachments[POP3_ATTACH_MAX][POP3_LINE_MAX];
```

Variables

The following are a list of i!-EquipmentMonitorIn.axi variables:

```
VOLATILE
_sEmailMessage
sEmailMessage[POP3_MAX_EMAILS]  Emails retrieved from server

VOLATILE
Integer
nPop3QtyMail;                   Number of message retrieved from server

VOLATILE
Integer
nPop3TotalMail;                 Number of total messages on server

VOLATILE
CHAR
bPop3Debug                      Set to 1 to debug
```

Functions

The following table lists i!EmailIn.axi functions.

i!-EquipmentMonitorIn.axi Functions	
Pop3ClearAllEmail Messages Deletes all e-mails from the internally stored email list.	Syntax: `Pop3ClearAllEmailMessages()` `Pop3ClearAllEmailMessages` has no arguments. Example: `Pop3ClearAllEmailMessages()` Remarks: `Pop3ClearAllEmailMessages` should be called when you want to delete all messages from the internally stored e-mail list. `Pop3ClearAllEmailMessages` updates `nPop3QtyMail` and `nPop3TotalMail` accordingly. This function does not delete e-mails from the server.
Pop3ClearEmail Message Deletes an e-mail from the internally stored e-mail list.	Syntax: `SLONG Pop3ClearEmailMessage(Integer MsgNum)` `Pop3ClearEmailMessage` has these arguments: MsgNum: The message number of the e-mail to be deleted. `Pop3ClearEmailMessage` returns these values: -1 and 0: If the MsgNum is invalid the e-mail was deleted successfully.
	Example: `Pop3ClearEmailMessage(1)` Remarks: `Pop3ClearEmailMessage` should be called when you want to delete a message from the internally stored e-mail list. Deleting an e-mail from the internally stored list will most likely affect message ordering. `Pop3ClearEmailMessage` updates `nPop3QtyMail` and `nPop3TotalMail` accordingly. This function does not delete an e-mail from the server.

Pop3GetEmail
Retrieves e-mail from the server.
	Syntax:
	`Pop3GetEmail(CHAR Delete)`
	`Pop3GetEmail` has these arguments:
	Delete: 1 or 0. 1 will delete all e-mails from the server as they are retrieved; 0 will leave all e-mails on the server.
	Example:
	`Pop3GetEmail(1)`
	Remarks:
	`Pop3GetEmail` should be called when you want to manually force the retrieval of e-mail from the server. By default, it is not retrieved from the server automatically, and calling `Pop3GetEmail` is the only way to retrieve e-mail. If you have called `Pop3SetRefresh` to enable automatical e-mail retrieval, calling `Pop3GetEmail` also resets the timer so e-mail will not be retrieved again until the current refresh time has expired.

i!-EquipmentMonitorIn.axi Functions (Cont.)	

Pop3SetRefresh
Sets the refresh time the include file checks for new e-mails and whether they should be deleted form the server.
	Syntax:
	`Pop3SetRefresh(Integer Refresh, CHAR Delete)`
	`Pop3SetRefresh` has these arguments:
	Refresh: Integer containing the refresh time in seconds. 0 disables automatic e-mail retrieval.
	Delete: 1 or 0. 1 will delete all e-mails from the server as they are retrieved; 0 will leave all e-mails on the server.
	Example:
	`Pop3SetRefresh(120,1)`
	Remarks:
	`Pop3SetRefresh` should be called if you want to adjust when and how e-mail is automatically retrieved from the server. By default, e-mail is not retrieved from the server automatically.

Pop3SetServer
Sets Your POP3 Server Name for your use.
	Syntax:
	`Pop3SetServer(CHAR Server[])`
	`Pop3SetServer` has these arguments:
	Server String containing the name or IP of your e-mail (POP3) server.
	Example:
	`Pop3SetServer('mail.amx.com')`
	Remarks:
	`Pop3SetServer` should be called in `DEFINE_START` of your application.

Pop3SetUser
Sets your POP3 user name and password for you e-mail (POP3) account.
	Syntax:
	`Pop3SetUser(CHAR User[], CHAR Pass[])`
	`Pop3SetUser` has these arguments:
	User String containing the user name of your e-mail (POP3) account.
	Pass String containing the user password of your e-mail (POP3) account.
	`Pop3SetUser` does not return a value.
	Example:
	`Pop3SetUser('vmorrison','GoldenAutumnDay')`
	Remarks:
	`Pop3SetUser` should be called in `DEFINE_START` of your application.

AMX reserves the right to alter specifications without notice at any time.

brussels • dallas • los angeles • mexico city • philadelphia • shanghai • singapore • tampa • toronto* • york
3000 research drive, richardson, TX 75082 USA • 469.624.8000 • 800.222.0193 • fax 469.624.7153 • technical support 800.932.6993